

On Constructing Efficient Shared Decision Trees for Multiple Packet Filters

Bo Zhang T. S. Eugene Ng
Department of Computer Science
Rice University

Abstract—Multiple packet filters serving different purposes (e.g., firewalling, QoS) and different virtual routers are often deployed on a single physical router. The HyperCuts decision tree is one efficient data structure for performing packet filter matching in software. Constructing a separate HyperCuts decision tree for each packet filter is not memory efficient. A natural alternative is to construct shared HyperCuts decision trees to more efficiently support multiple packet filters. However, we experimentally show that naively classifying packet filters into shared HyperCuts decision trees may significantly increase the memory consumption and the height of the trees. To help decide which subset of packet filters should share a HyperCuts decision tree, we first identify a number of important factors that collectively impact the efficiency of the resulted shared HyperCuts decision tree. Based on the identified factors, we then propose to use machine learning techniques to predict whether any pair of packet filters should share a tree. Given the pair-wise prediction matrix, a greedy heuristic algorithm is used to classify packets filters into a number of shared HyperCuts decision trees. Our experiments using both real packets filters and synthetic packet filters show that the shared HyperCuts decision trees consume considerably less memory.

Index Terms—Packet filter, packet classification, data structure, virtual router, HyperCuts.

I. INTRODUCTION

Packet filters are widely used on network devices such as routers to perform various services including firewalling, quality of service (QoS), virtual private networks (VPNs), load balancing, traffic engineering, etc. Therefore, multiple packet filters serving different purposes may be deployed on a single physical router. With the emergence of virtual routers as a promising technology to provide network services, even more packet filters belonging to different virtual routers need to be stored on a single physical host router. In this paper, we show that by using a shared data structure to represent multiple packet filters, memory consumption can be considerably reduced. Consequently, more packet filters can be deployed and more virtual routers can be efficiently supported on a single physical router.

A. The Need for Multiple Packet Filters on a Single Router

Multiple packet filters may be deployed on a single router to support different network services such as firewalling, QoS, VPNs, load balancing and traffic engineering. Due to the complexity of the network services, each packet filter may be large and complex as well. For example, recent studies have shown that a complex packet filter on modern routers or firewalls can have as many as 50,000 rules [24].

Today router virtualization is already available in commercial routers from both Cisco [1] and Juniper [2]. It is quickly emerging as a promising technology to support new network services such as router consolidation [12], customer-specific routing, policy-based routing [7], multi-topology routing [17] [20] and network virtualization [3] [5]. For example, with the help of router virtualization, network operators can now consolidate a large number of existing routers onto a newly-purchased router by running one virtual router instance for each existing router. When performing router consolidation, all the packet filters deployed on existing routers will be exported to the new router. A Juniper router today can be configured with as many as 128 virtual routers. Therefore, a modern router may need to support a large number of packet filters.

B. Challenges of Deploying Multiple Packet Filters on a Single Router

One key challenge of holding a large number of packet filters on a single physical router is memory consumption. As more packet filters are deployed, the memory requirement will also increase accordingly.

Ternary content addressable memory (TCAM) is the de facto industry standard for hardware-based fast packet classification. However, TCAM has a few limitations. Firstly, TCAM consumes lots of power. Secondly, TCAM chips are expensive. They are often more expensive than network processors [13]. Thirdly, due to its high power consumption and high cost, the capacity of TCAM on each router is usually restricted by system designers. What is worse, in order to represent a packet filter in TCAM, the packet filter rules have to be converted to the ternary format, which will lead to the range expansion problem. For example, the Cisco 12000, a high-end Gigabit switch router designed for large service provider and enterprise networks, can only hold up to 20,000 rules in its TCAM. Although some recently proposed TCAM-based packet classifier compression techniques [15] [6] may help to alleviate this problem, the amount of memory required to store a large number of packet filters can still easily exceed the capacity of the installed TCAM on a physical router.

Therefore, software based packet classification using fast memory such as SRAM is still widely used on many routers including both edge routers such as the Cisco 7200 series and core routers such as the Cisco 12000 series. Although SRAM consumes less power and occupies smaller space, it

is still costly. Therefore, any technique that can considerably reduce the memory requirement of holding multiple packet filters can be useful in practice. The saved memory can be used to improve cache performance, to more efficiently hold more packet filters and to support more virtual routers.

C. Improving Memory Efficiency by Sharing Data Structure

In software based packet classification systems, each packet filter is represented by a data structure such as a decision tree. A separate data structure for each packet filter is not memory efficient. A natural alternative is to use a shared data structure. In this paper, we will use the HyperCuts [18] decision tree to represent packet filters since it is one of the most efficient data structures for performing packet filter matching.

In Section II, we briefly introduce the HyperCuts data structure and then extend the original HyperCuts data structure to support multiple packet filters. Section III first uses a simple experiment to show that naively clustering packet filters to shared HyperCuts decision trees may result in significantly increased memory consumption. Section IV presents our approach of clustering packet filters into multiple shared HyperCuts decision trees. The idea is to first identify important factors that can affect the efficiency of the constructed shared HyperCuts decision tree. Based on the identified factors, we then leverage machine learning techniques to predict which pairs of packet filters should share a tree. Given the pair-wise prediction, a heuristic clustering algorithm is used to cluster all packet filters into a number of shared HyperCuts decision trees. We evaluate the accuracy of the pair-wise prediction and the memory saving by constructing shared trees for clustered packet filters in Section V. Evaluation results using both real packet filters and synthetic packet filters show that the pair-wise prediction is accurate and the shared HyperCuts decision tree can considerably reduce the memory consumption. We discuss related work in Section VI and conclude in Section VII.

II. BACKGROUND

A. Packet Filters Notations

Informally, a packet filter of size n is a list of n ordered rules $\{R_1, R_2, \dots, R_n\}$ that collectively define a packet classification policy. Each rule R_i is composed of two parts: a combination of D values, one for each selected packet header field, and an associated action. The most commonly used five packet header fields are: source IP address, destination IP address, source port, destination port, and protocol type. Each of the D values specified in R_i could be a single value or an interval of values or the special value ANY used to specify all possible legitimate values for that field. Typical actions associated with a rule include permit, deny, marking the ToS bit, etc. A packet P is considered to match the rule R_i if all the D header fields of P match the corresponding values in R_i . If P matches more than one rule, then the rule with the smallest index in the packet filter is returned. The associated action of the returned rule will be performed on P accordingly.

A simple packet filter with 10 rules defined on 5 fields is shown in Table I.

B. The HyperCuts Data Structure and Algorithm

Decision trees have been shown to be a powerful data structure for performing packet classification by using geometric cutting [21]. Several different variants of decision tree based packet classification algorithms (e.g., [23] [11] [18]) have been proposed. HyperCuts [18] is considered to be one of the most efficient decision tree based algorithms. In this section, we will briefly introduce the HyperCuts data structure and algorithm. A more detailed discussion can be found in [18].

A HyperCuts decision tree is composed of two types of nodes: internal nodes and leaf nodes. Each leaf node contains less than $BucketSize$ number of rules, where $BucketSize$ is a small constant (e.g., 4). The small number of rules stored in a leaf node will be linearly traversed to find the matched rule with the smallest index in the original packet filter. By contrast, an internal node contains more than $BucketSize$ rules, so rules stored in the internal node have to further split to its child nodes.

The HyperCuts decision tree is efficient because it splits rules in internal nodes using the information from multiple packet fields. In contrast to HyperCuts, HiCuts [11] only splits rules on one packet field at a time. In order to decide which subset of packet fields to use to split rules on an internal node, the HyperCuts algorithm will first count the number of unique elements on each field for all rules stored on the node. Let us take the 10 rules in Table I as an example, the number of unique elements in all five fields is 10, 10, 1, 9, 2 respectively. The HyperCuts algorithm will then consider the set of fields for which the number of unique elements is *greater than the mean* of the number of unique elements for all the fields. For example, given a node holding the 10 rules in Table I, the three fields of source IP, destination IP and destination port should be considered for cutting. After determining which set of fields to cut, the HyperCut algorithm uses several heuristics to decide how many cuts should be performed on each field. Due to the space limitation, we will not discuss those heuristics in detail here. However, it is worth noting that the number of child nodes that an internal node can be split into is limited by a factor of the number of rules stored in the node. The function is defined as $f(N) = spfac \times \sqrt{N}$, where N is the number of rules in the internal node and $spfac$ is a small constant with a default value of 2. This technique is used by both the HiCuts and the HyperCuts algorithms to reduce the memory consumption.

C. Extend the HyperCuts Data Structure and Algorithm

To allow multiple packet filters to share a HyperCuts tree, the original HyperCuts data structure and tree building algorithm need to be extended. Figure 1 (a) shows two separate HyperCuts trees, each of which only has one internal node (its root) and four leaf nodes. Figure 1 (b) shows the corresponding shared HyperCuts tree. As can be seen, the internal node on shared HyperCuts tree is the same as the one in the original

Rule ID	Source IP	Destination IP	Source port	Destination port	Protocol	Action
R_0	104.253.26.143/31	151.217.12.0/23	ANY	1489	TCP	$act0$
R_1	103.11.193.196/31	151.193.40.150/32	ANY	27000	TCP	$act0$
R_2	51.109.218.92/30	243.82.86.0/23	ANY	135	TCP	$act1$
R_3	133.202.88.44/30	78.87.20.226/31	ANY	[1300-1349]	TCP	$act2$
R_4	137.180.89.7/32	243.82.125.14/32	ANY	6789	TCP	$act1$
R_5	201.130.210.90/31	6.92.31.0/25	ANY	1533	TCP	$act0$
R_6	119.10.210.90/31	6.92.31.0/25	ANY	1526	UDP	$act0$
R_7	119.67.166.172/31	151.143.84.75/32	ANY	1521	TCP	$act3$
R_8	71.252.162.33/32	151.166.64.162/32	ANY	[1300-1349]	TCP	$act4$
R_9	209.137.112.252/31	151.248.122.158/32	ANY	[61200-61209]	TCP	$act2$

TABLE I
A SIMPLE PACKET FILTER EXAMPLE WITH 10 RULES ON FIVE FIELDS

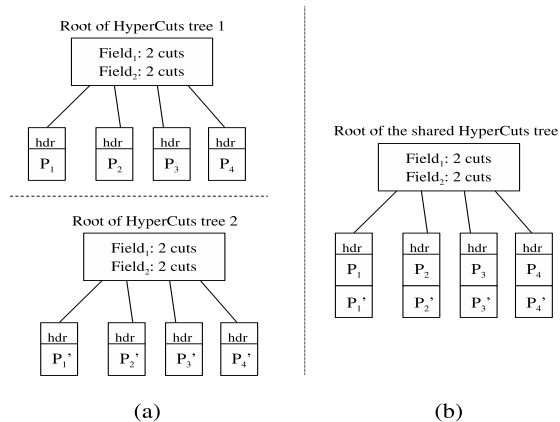


Fig. 1. Example of Shared HyperCuts tree: (a) Two separate HyperCuts trees. (b) The corresponding shared HyperCuts tree.

HyperCuts tree. Each internal node only records the number of cuts performed on each field and a list of pointers to its child nodes. On the other hand, leaf nodes have to be slightly extended to support multiple packet filters sharing the tree. In the original HyperCuts tree, a leaf node is composed of a header (indicating the node is a leaf node) and a pointer to the set of rules stored in this leaf node. In the shared HyperCuts tree of *two* packet filters, a leaf node is composed of the same header and *two* pointers, one for each packet filter. When a packet reaches a leaf node when searching the shared HyperCuts tree, since it knows which packet filter this packet is being matched, it will directly calculate which pointer it should access next. Therefore, the time to access a leaf node on the shared HyperCuts tree is still the same as in the original HyperCuts tree. In this simple example, by making the two packet filters share a tree, we saved one internal node and 4 headers of leaf nodes.

Now we continue to explain how we extend the original HyperCuts tree construction algorithm. The idea is to use a corresponding average value across all packet filters to replace the value used in the original algorithm. For example, suppose that the two packet filters F_1 and F_2 are sharing a HyperCuts decision tree. Given an internal node on the shared tree, if the number of stored rules from each packet filter is N_1 and N_2 , then the number of child nodes this internal node can have is bounded by $spf_{ac} \times \sqrt{(N_1 + N_2)/2}$. Similarly, to decide the subset of fields for cutting on each internal node, we will first calculate the number of unique elements in each field on a per packet filter basis. Let us denote the number of unique

elements for rules from F_1 and F_2 as u_{1j} and u_{2j} respectively, where $1 \leq j \leq D$. Then the number of unique elements on each field u_j for the current internal node is defined as $u_j = (u_{1j} + u_{2j})/2$. The rest of the algorithm is just the same as original HyperCuts algorithm.

D. Efficiency Metrics of The HyperCuts Decision Tree

Given a constructed HyperCuts tree, we wish it consumes as little memory as possible. Thus, a natural metric of interest is **memory consumption**. In addition, we wish to do fast packet classification using the shared HyperCuts tree, so the tree search time (i.e., from the root to leaf nodes) is also important. We use the following two metrics to characterize the tree search time:

Average depth of leaf nodes: The depth of a leaf node is just the length of the shortest path from itself to the root. Assuming each leaf node has the same probability to be reached during a packet matching, then the average depth of all leaf nodes reflects on average how many internal nodes need to be accessed to terminate this tree search.

Height of the tree: This metric characterizes the largest number of internal nodes needed to be accessed for a packet to reach a leaf node. It corresponds to the worst case search time.

III. CHALLENGES OF CONSTRUCTING EFFICIENT SHARED HYPERCUTS DECISION TREE

To construct efficient shared HyperCuts decision trees, one key question to answer is: which subset of packet filters should share a HyperCuts decision tree so that the resulted shared tree is more efficient than a set of separate trees? In this section, we first introduce the filter data sets used in the paper. We then experimentally show that naively letting multiple packet filters share a HyperCuts decision tree will significantly increase the memory consumption and height of the shared trees.

A. Filter Data Sets

We extracted a set of real packet filters from the configuration files of routers in a large-scale campus network [19] at Purdue University. We did not include the 260 packet filters that contain no more than *BucketSize* number of rules, because their corresponding HyperCuts decision trees just contain one root node. In our experiment throughout the paper, we always set *BucketSize* as 4.

Because it is hard to obtain other real packet filters, a synthetic filter generator ClassBench [22] is used to generate some synthetic filters. The ClassBench tool takes a parameter file as the input and then generate synthetic filters using the information stored in the input parameter file. We used three parameter files provided by ClassBench and they were originally generated from three real access control lists (ACLs) on Cisco routers. Given each parameter file, we generate two sets of 1,000 synthetic filters. The first set of 1,000 synthetic filters all contain 100 rules, while the size distribution of the second set of 1,000 synthetic filters follows an exponential distribution with the average value of 100. Please note that when generating synthetic filters with exponential size distribution, we also discard the filters containing no more than *BucketSize* rules.

Some basic statistics about the set of real packet filters and the six sets of synthetic filters are summarized in Table II.

B. Making Random Packet Filters Share HyperCuts Trees?

In this section, we will use a simple experiment to show that extra care has to be taken in deciding which set of packet filters should share a tree. Naively making a set of random packet filters share a tree will significantly degrade the performance.

In our experiment, for each filter data set, we randomly choose n distinct filters, where n is a small number. Given the n selected filters, we first build a separate tree for each selected filter. Let us denote the memory consumption of the n trees as m_i , the average depths of leaf nodes of the n trees as d_i , and the heights of the n trees as h_i , where $1 \leq i \leq n$. Then we construct a shared HyperCuts decision tree to represent the selected n filters. Let us denote the memory of the shared tree, the average depth of leaf nodes in the shared tree and the height of the shared tree as m_{shared} , d_{shared} and h_{shared} . Now we can define the **memory consumption ratio** as $m_{shared}/\sum_{i=1}^n m_i$, the **average leaf depth ratio** as $d_{shared}/(\sum_{i=1}^n d_i/n)$, and the **tree height ratio** as $h_{shared}/(\sum_{i=1}^n h_i/n)$. The smaller the ratios are, the more benefits we obtain by making the n packet filters share a single HyperCuts tree. A ratio larger than 1 means that the shared tree has worse performance than n separate trees. Given each fixed n , we repeat the experiment 1000 times, i.e., we randomly select 1000 sets of n distinct filters for our experiment. We also vary n from 2 to 10.

Figure 2 (a) shows the average memory consumption ratio across 1000 runs for all 7 data sets. As can be seen, when the number of randomly selected filters increases, the memory consumption ratio becomes higher for all 7 data sets. This is because the more random packet filters are selected, the harder it is to construct a HyperCuts tree suitable for all packet filters. When 10 random packet filters share a HyperCuts decision tree, it will consume 2 to 20 times more memory than simply using 10 separate trees. Figure 2 (b) shows the average of the average leaf depth ratios across 1000 runs. Similarly, the more random packet filters are selected to share a tree, the larger the ratios are. The tree height ratio results are very similar to the average leaf depth ratio results, so they are not shown here.

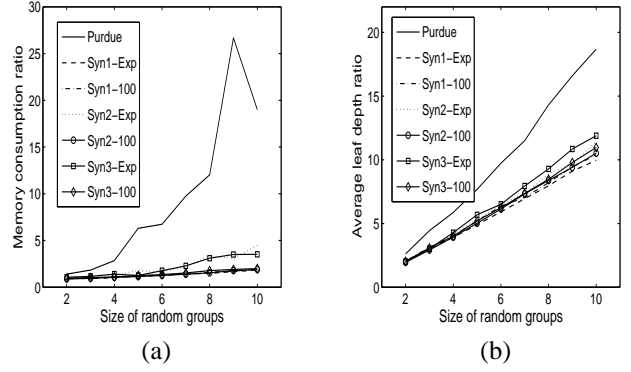


Fig. 2. (a) Memory consumption increases when random packet filters share a HyperCuts tree. (b) Average depths of leaf nodes increase when random packet filters share a HyperCuts tree.

By comparing the memory consumption ratio and average leaf depth ratio, we can also observe that the average leaf depth ratio increases more rapidly with the increase of n than the memory consumption ratio does. The reason is that the sizes of all internal nodes in a HyperCuts tree are not the same. Please recall that the number of child nodes that an internal node can have is related to the number of rules stored in the node. Because those nodes closer to the root usually contain more rules, they accordingly have more child pointers (4 bytes for each pointer). Thus, internal nodes closer to the root are much larger than the internal nodes far from the root. This explains why a HyperCuts tree with doubled height consumes less than doubled memory.

IV. CLUSTERING PACKET FILTERS TO CONSTRUCT EFFICIENT SHARED HYPERCUTS DECISION TREES

As shown in Section III, letting a set of random filters share a HyperCuts tree leads to increased memory consumption and tree search time. In this section, we propose a novel approach to clustering packet filters to form efficient shared HyperCuts decision trees. In our approach, to help decide which subset of packet filters should share a tree, we first identify a number of important factors that collectively impact the efficiency of the resulted shared tree. Based on the identified factors, we then propose to use machine learning techniques to predict whether any pair of packet filters should share a HyperCuts decision tree. Given the pair-wise prediction on all possible pairs, a greedy heuristic algorithm is used to classify packets filters into a number of shared HyperCuts decision trees.

A. Factors Affecting the Efficiency of the Shared Tree

In this section, we first present some important factors that can characterize each individual packet filter. Then based on the collection of factors from a set of packet filters, we can decide whether they should share a tree or not. According to our analysis, there are two classes of factors that can impact the efficiency of the shared HyperCuts decision tree:

Class-1 factors include some simple statistical properties of a packet filter itself. They include the *size of the packet filter* and the *number of unique elements in each field*. As you can see, to obtain the Class-1 factors, we do not need

Data Set Name	Parameter File	Number of Filters	Size Distribution	Average Size	Minimum Size	Maximum Size
Purdue	N/A	140	N/A	21.5	5	763
Syn1-Exp	ACL1	1000	Exponential	98.21	5	1002
Syn1-100	ACL1	1000	Uniform size:100	100	100	100
Syn2-Exp	ACL3	1000	Exponential	101.9	5	910
Syn2-100	ACL3	1000	Uniform size:100	100	100	100
Syn3-Exp	ACL4	1000	Exponential	106.3	5	874
Syn3-100	ACL4	1000	Uniform size:100	100	100	100

TABLE II
SUMMARY OF BASIC STATISTICS ABOUT THE SEVEN FILTER DATA SETS.

to build the HyperCuts decision tree for the packet filter. These factors are important because they are used in the HyperCuts tree construction algorithm. Thus, they can affect the structure of the final HyperCuts tree. However, only Class-1 factors are not enough to determine the structure or memory consumption of the final HyperCuts decision tree. Two packet filters with identical Class-1 factors may have very different tree structures. Therefore, we identify the second class of factors as follows.

Class-2 factors represent the characteristics of the constructed HyperCuts decision tree. That is, the HyperCuts tree must be constructed to obtain the Class-2 factors of a packet filter. Because we want the final shared tree to have good performance, the *memory consumption of the tree*, the *average depth of leaf nodes* and the *height of the tree* are one part of the Class-2 factors. In addition, the *number of leaf nodes*, the *number of internal nodes* and the *total number of cuts on each field* are also included into the Class-2 factors, because they can more accurately reflect the actual structure and memory consumption of the HyperCuts tree. For example, the more nodes a tree has, the more memory it will generally consume. In addition, the total number of cuts performed on each field can reflect the relative importance of each field so it can impact the structure of the constructed tree.

Given the two classes of important factors, now we may cluster all the packet filters into a number of shared trees using their corresponding factors. To make the packet filters clustering problem simpler, in the following section, we will first study how to determine whether two packet filters should share a tree using their corresponding factors.

B. Predicting Good Pairs of Packet Filters

Two packet filters are defined to be a “good” pair if their shared HyperCuts tree has decreased memory usage and decreased average depth of leaf nodes compared to the two separate HyperCuts trees. This problem is clearly a classification problem, i.e., we need to classify all pairs of packet filters into either good pairs or bad pairs. However, it is non-trivial to manually derive some effective rules for us to accurately decide whether a pair of packet filters should share a tree or not. Luckily, some effective supervised machine learning techniques [16] can help perform this classification task. We will study a few representative supervised machine learning techniques in Section V.

To use machine learning techniques to predict whether a pair of filters is good, we need to first prepare some training data to train a model. Given a filter data set with N distinct packet filters, we can randomly select M pairs of filters out of

all possible $N \times (N - 1) / 2$ pairs as the training data. For each selected pair of filters, we can decide whether they are a good pair by constructing two separate trees and one shared tree. For each selected pair of filters, we can also calculate their factors. By feeding all these information to certain machine learning technique, a model can be learned to be used to predict whether any new pair of packet filter is good or bad. We will evaluate the prediction accuracy of different machine learning techniques in Section V.

C. Clustering Packet Filters Based on Pair-wise Prediction

By using the model learned from a small amount of training pairs, we can now predict whether any pair of filters is good or not. Based on the pair-wise prediction for all possible pairs of all filters, an undirected graph G can be constructed. In the graph G , each node represents a distinct packet filter. Two nodes in G are connected with an edge if and only if the two corresponding packet filters are predicted to be a good pair. Given the constructed graph G , the following clustering algorithm is proposed to determine which subset of packet filters should share a HyperCuts decision tree:

INPUT OF ALGORITHM: G and $\alpha \in [0, 1]$

OUTPUT OF ALGORITHM: A set of packet filter clusters: $S_{clusters}$

```

01:  $S_{filters} = \{\text{All packet filters}\}$ ;
02:  $S_{clusters} = \{\}$ ;
03: WHILE ( $|S_{filters}| > 0$ )
04:    $cluster_i = \{\}$ ;
05:   Find  $f_m \in S_{filters}$  who has most neighbors from  $S_{filters}$  in  $G$ ;
06:    $cluster_i = cluster_i \cup \{f_m\}$ ;
07:    $S_{filters} = S_{filters} \setminus \{f_m\}$ ;
08:   WHILE TRUE
09:     Find  $f_n \in S_{filters}$  with most neighbors from  $cluster_i$  in  $G$ 
      if multiple choices exist, pick the one with largest degree in  $G$ ,
      let us assume  $f_n$  has  $k$  neighbors from  $cluster_i$  in  $G$ ,
10:     IF ( $k \geq \alpha \times |cluster_i|$ )
11:        $cluster_i = cluster_i \cup \{f_n\}$ ;
12:        $S_{filters} = S_{filters} \setminus \{f_n\}$ ;
13:     ELSE break the WHILE loop;
14:   END-IF
15: END-WHILE
16:  $S_{clusters} = S_{clusters} \cup \{cluster_i\}$ ;
17: END-WHILE
18: RETURN  $S_{clusters}$ ;

```

In the above algorithm, α is a constant value between 0 and 1. Intuitively, the higher the α value is, the more difficult that a packet filter can join an existing cluster. For example, if α is set to 0, then all packet filters in the same connected component in G will share a HyperCuts decision tree. On the other hand, if α is set to 1, then a set of packet filters will be clustered together if and only if the corresponding nodes in G form a clique. We will evaluate the performance of the clustering algorithm with different α values in Section V-B.

V. PERFORMANCE EVALUATION

In Section V-A, we first evaluate how accurately we can predict whether a pair of packet filters should share a tree. We then study the performance of the packet filter clustering algorithm in Section V-B. Finally, we show the detailed breakdown of the time spent on each step of our approach in Section V-C.

A. Accuracy of Predicting Good Pairs

As introduced in Section IV, we want to apply supervised machine learning techniques to address this classification problem. A supervised machine learning technique can automatically learn a model from some training data. The training data consists of pairs of input vectors, and desired outputs. After a model is learned, it can then be used to predict an output value for any valid input vectors. We discuss how we define the input vectors, the output values and three classification techniques we studied in detail as follows.

1) *Three Types of Input Vectors*: Based on the two classes of factors introduced in Section IV-A, we can define three types of input vectors for each pair of packet filters. The first type of input vectors is composed of only the Class-1 factors from both filters. The second type of input vectors is composed of only the Class-2 factors of both filters. The third type of input vectors includes both the Class-1 and Class-2 factors of the two filters. We evaluate the impact of using different types of input vectors in Section V-A4.

2) *Defining Output Values*: The output of our classification problem should be a label indicating whether the input vectors correspond to a good pair or not. That is, there are only two possible output values: good or bad. In this section, we define two packet filters as a good pair if their shared HyperCuts tree’s memory consumption ratio and average leaf depth ratio are both smaller than 1. That is, the shared HyperCuts tree must have decreased memory consumption and decreased average depth of leaf nodes compared against two separate HyperCuts trees. Please note that in the above definition, if we replace the average depth of leaf nodes with the height of the tree, the prediction accuracy is a little worse according to our study. The reason is that the heights of trees are determined by the leaf node with largest depth, so it is not as stable as the average depth of all the leaf nodes. Due to limited space, we only present the prediction accuracy by using the definition of good pairs based on average leaf depth ratio and memory consumption ratio.

We studied the percentage of good pairs by examining 10,000 random selected pairs from each data set. The fractions of good pairs vary from 8% to 16% across all 7 data sets. Since the fractions of good pairs are relatively small, any classification technique that can accurately identify good pairs can be useful in practice.

3) *Three Classification Techniques*: We studied three representative classification techniques including the decision tree

(DT) [16]¹, the generalized linear regression (GLR) [16] and the naive Bayes classifier (NBC) [16]. We plan to study more classification techniques such as the neural network in the future.

It is straightforward to apply the DT technique to perform classification here. For GLR technique, if we use the output values 1 and 0 to represent the good pair and the bad pair respectively in the training data, then given a new pair of filters, GLR will output a value between 0 and 1. In our experiment, if the returned value by GLR is larger than 0.5 then we predict the pair as good. Otherwise, the pair is predicted to be bad. As for NBC, we cannot directly feed the input vectors defined in Section V-A1 to NBC technique. NBC requires a set of features instead. In our experiment, we simply define a corresponding feature from each factor. For example, the size of the first packet filter in the pair is a factor. We can define its corresponding feature as follows: we first calculate the 10th percentile and 90th percentile of the sizes of the first packet filter from all good pairs in the training data. A pair of testing packet filters is then said to have this feature if the size of its first packet filter falls into the above 10th and 90th percentile range. After we convert factors into features, the NBC can be used directly to perform classification.

4) *Accuracy of Pair-Wise Prediction*: For each data set, we randomly select 10,000 pairs and then calculate both Class-1 and Class-2 factors for those selected pairs. We also need to determine whether each selected pair is good or bad. To evaluate the prediction accuracy using different types of input vectors, we randomly choose 1,000 pairs (i.e., 10%) out of the 10,000 pairs as the training data. We then use the rest 9,000 pairs as the testing data to test the prediction accuracy of the learned model. We repeat this experiment 10 times, each of which uses a different 1,000 pairs as the training data. Figure 3 and 4 show the average false positive rate and the average false negative rate of the three classification techniques using different input vectors across 10 runs.

First of all, different types of input vectors have a significant impact on the false positive and false negative rate for all three techniques. Only using Class-1 factors as input gives the worst prediction accuracy for both DT and GLR. Including Class-2 factors in the input vectors help improve the performance of both DT and GLR. This is expected because Class-1 factors are relatively simple and they are not sufficient to predict the final HyperCuts decision tree. However, including more factors as input does not help NBC. Instead, when more and more factors are included as input, the performance of NBC is getting worse. The NBC technique assumes that all the input variables are independent to each other, while in our case, those input factors may not be completely independent. When having more and more dependent variables into the input vectors, the performance may get worse.

Secondly, among the three techniques we have studied, DT technique has the best overall performance. GLR does not

¹To avoid ambiguity, we always use “HyperCuts decision tree” to refer the packet classification technique, while using “decision tree” or “DT” in this section to represent the machine learning techniques used

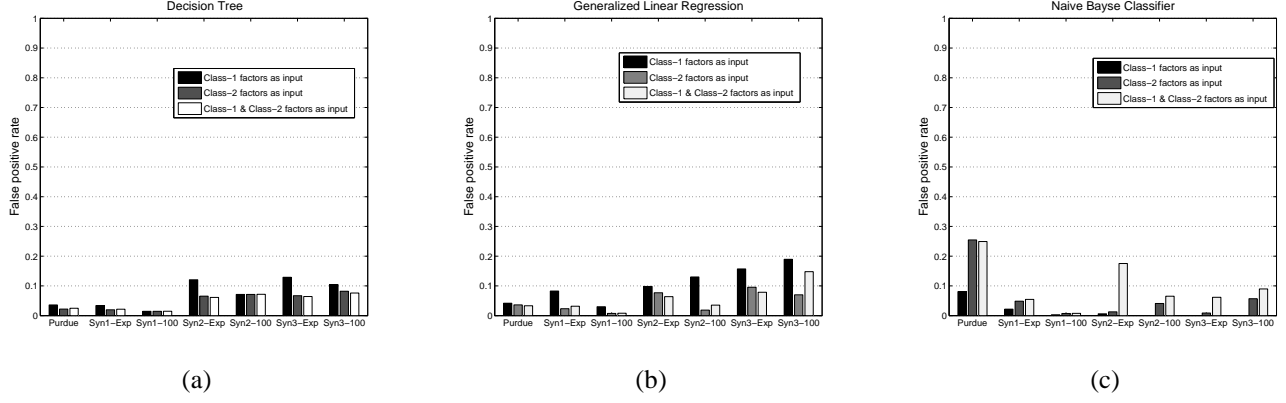


Fig. 3. False positive rate: (a) Decision tree (DT) (b) Generalized linear regression (c) Naive Bayes classifier.

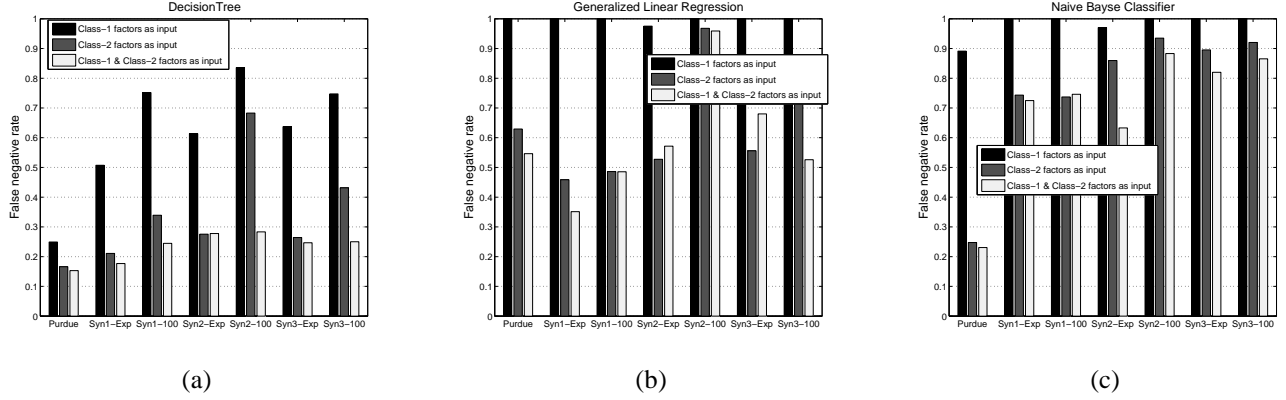


Fig. 4. False negative rate: (a) Decision tree (DT) (b) Generalized linear regression (c) Naive Bayes classifier.

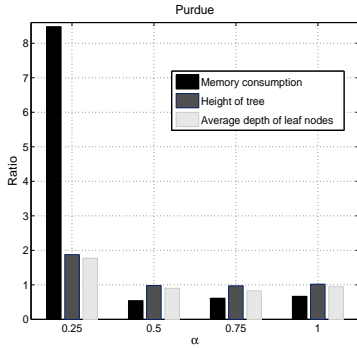


Fig. 5. Shared HyperCuts trees V.S. separate HyperCuts trees (Purdue data).

work well because its linear model simply can not accurately capture the complex relationships among those factors. NBC falls short because it assumes that all factors are independent while they are actually not. If both Class-1 and Class-2 factors are used in the input vectors to train the decision tree, then the false positive rates will vary from 3% to 8%. In addition, the average false negative rate across the 7 data sets is 23%. A low false positive rate is important because it means that only a small percentage of bad pairs will be mistakenly predicted to be good ones. A 23% false negative rate means that 77% of all good pairs can still be correctly identified by the learned model.

B. Performance of The Filter Clustering Algorithm

Since we have shown that the DT technique using both Class-1 and Class-2 factors as input has the best prediction accuracy among the three techniques we studied, in this section we will use DT to predict the goodness of all pairs of packet filters in a data set. Based on the pair-wise prediction provided by DT, we can construct a graph G for each filter data set. We can then apply our filter clustering algorithm to cluster nodes in G to decide which subset of packet filters should share a HyperCuts decision tree. DT is trained by using a training data set of 1,000 random pairs, and the results presented in this section are the average values across 10 runs. Recall that in addition to G , the proposed clustering heuristic algorithm also needs a constant α . In our experiment, we vary α from 0.25 to 1.

Figure 5 shows the performance of the final constructed shared trees for 140 Purdue filters. When $\alpha = 0.25$, the shared trees actually have much worse performance than the 140 separate trees. Please recall that a smaller α value means that a packet filter can more easily join an existing cluster. When a packet filter is mistakenly classified into a wrong filter cluster, the overall performance of the cluster will significantly degrade. When a larger α such as 0.5 is used, the performance becomes better. The overall memory saving is over 40%. In the meantime, the average heights of the shared trees and the average depth of leaf nodes in shared trees are slightly decreased.

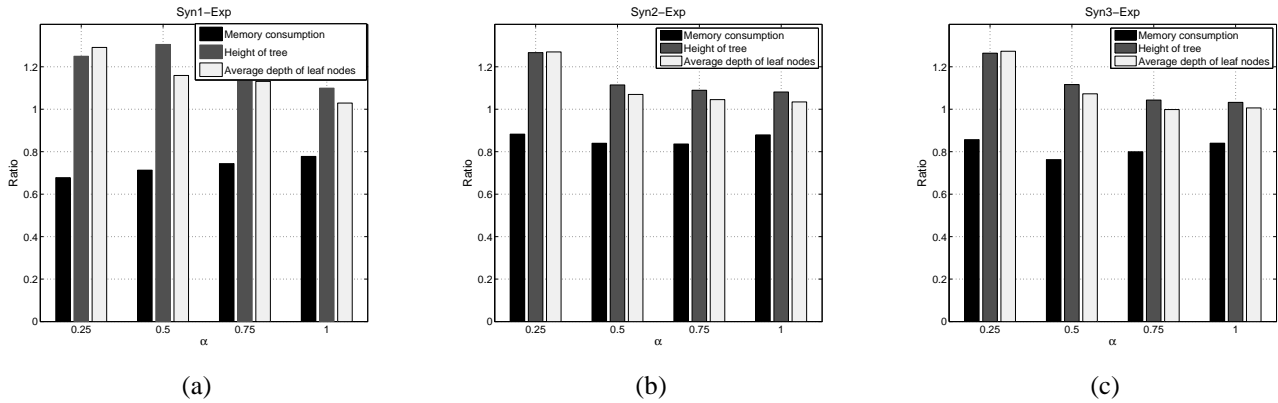


Fig. 6. Shared HyperCuts trees V.S. separate HyperCuts trees: (a) Syn1-Exp (b) Syn2-Exp (c) Syn3-Exp.

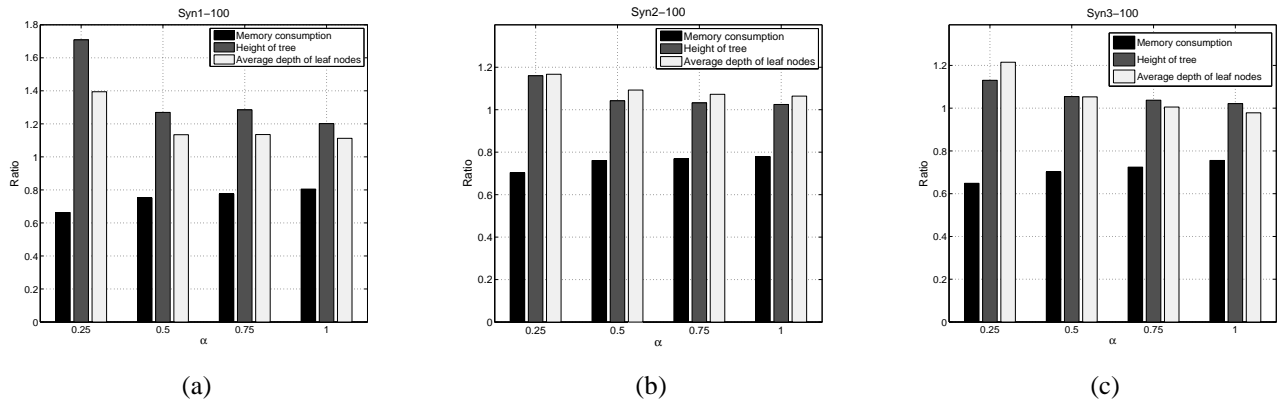


Fig. 7. Shared HyperCuts trees V.S. separate HyperCuts trees: (a) Syn1-100 (b) Syn2-100 (c) Syn3-100.

Figure 6 and Figure 7 show the overall performance of the 6 synthetic data sets. As can be observed, when α increases, the memory consumption ratio generally increases while the average leaf depth ratio and tree height ratio decrease. If we fix α as 1, then we can reduce memory consumption over 20% on average while only increasing average leaf depth by 3% on average across all 6 synthetic data sets.

As you can see, the parameter α plays a vital role in determining the filter clustering results and also the performance of the constructed shared HyperCuts decision trees. However, determining the optimal α value for a specific packet filter data set is beyond the scope of this paper. We will continue to study this problem as our future work.

C. Computation Time Breakdown

In this section, we want to study the computation time spent on each step in our approach. We break our approach into 7 steps: (1) calculating Class-1 factors, (2) calculating Class-2 factors, (3) generating 1,000 training pairs, (4) training the DT, (5) predicting the goodness of all pairs to construct G , (6) clustering packet filters and (7) constructing the shared HyperCuts decision trees. As for implementation, steps (4)-(6) are implemented in Matlab and the other steps are implemented in the C++ language. The desktop machine used in our experiment has a 2.6 GHz AMD Opteron processor and 4 GB of main memory.

Table III shows the detailed breakdown of time (in seconds)

spent on each step for all 7 data sets. When performing the packet filter clustering step, we set $\alpha = 0.5$. As can be seen, the step of preparing the training data takes the most time for all 7 data sets. The reason is that we need to construct 2,000 separate HyperCuts tree and 1,000 shared HyperCuts trees. The time spent in clustering packet filters and constructing shared HyperCuts trees is relatively modest. Therefore, a network operator may want to run the filter clustering and shared tree construction steps a few times with different α values to select one α offering best performance. In summary, it takes our approach about 17 seconds to construct shared HyperCuts trees for 140 real packet filters and about 6.8 minutes on average to construct a set of shared HyperCuts trees for 1,000 synthetic packet filters.

VI. RELATED WORK

To the best of our knowledge, this paper is the first to study how to construct efficient shared data structures for multiple packet filters. The HyperCuts [18] decision tree is used in our study because it is one of the most efficient packet classification data structures.

Our work is inspired by Fu and Rexford [9], who observed that the forwarding information bases (FIBs) of different virtual routers on the same physical router share a large number of common prefixes. They proposed to use a shared trie data structure to hold multiple FIBs. They also proposed a corresponding lookup algorithm to search the shared trie

Data Sets Name	Class 1	Class 2	Generating 1k training pairs	Training	Predicting G	Clustering	Constructing shared trees	Total
Purdue	1.7	4.9	6.9	0.29	0.01	0.52	2.6	16.92
Syn1-Exp	28.2	86.4	298.9	0.1	0.33	57.6	51.3	522.83
Syn1-100	24.7	69.3	138.6	0.3	0.7	62.0	21.1	316.7
Syn2-Exp	28.7	86.9	315.0	0.31	0.78	46.6	26.7	504.99
Syn2-100	23.5	74.5	143.9	0.15	0.33	7.7	48.6	298.68
Syn3-Exp	23.9	72.3	312.0	0.51	0.88	36.9	65.8	512.29
Syn3-100	24.5	74.2	141.1	0.41	0.75	31.7	35.3	307.96

TABLE III
COMPUTATION TIME BREAKDOWN (IN SECONDS) FOR EACH STEP IN THE PROPOSED APPROACH.

data structure. Their evaluation results show that by sharing a trie data structure, the memory requirement can be greatly reduced and the IP lookup time also decreases. However, their work only focused on merging forwarding tables. How to construct efficient shared data structures for multiple packet filters is not studied. In addition, in their approach, all FIBs are always merged into a single shared FIB, while our approach can automatically classify packet filters into multiple shared HyperCuts decision trees.

Several packet classifier compression techniques (e.g., [15], [6], [8], [4], [14]) for TCAM-based packet classification systems have been proposed. However, these techniques are specifically designed for optimizing TCAM-based systems. In addition, they all try to reduce TCAM memory usage by compressing each individual packet classifier, while the key idea of our approach is to save memory by allowing multiple packet filters to efficiently share data structures.

VII. CONCLUSION AND FUTURE WORK

To the best of our knowledge, this paper is the first to study how to construct efficient shared data structure such as the HyperCuts decision tree for multiple packet filters. We have identified a set of important factors that can affect the performance of the constructed shared HyperCuts trees. We then propose a novel approach to clustering packet filters into shared HyperCuts decision trees. Our evaluation using both real packet filters and synthetic packet filters shows that by enabling multiple packet filters to share HyperCuts decision trees, memory consumption can be considerably reduced. We also show that the proposed approach is practical. It only takes a few minutes to finish clustering 1,000 packet filters and to construct the corresponding shared HyperCuts decision trees. As future work, we will investigate how to efficiently cope with the dynamics of packet filters in practice. We plan to study efficient mechanisms for incrementally updating the shared decision trees when some packets filters are changed. We will also study whether our proposed technique can be applied to other data structures that can represent packet filters (e.g., the decision diagram [10]).

ACKNOWLEDGMENTS

This research was sponsored by the NSF under CAREER Award CNS-0448546, NeTS FIND Award CNS-0721990, by Microsoft Corp., and by an Alfred P. Sloan Research Fellowship. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied,

of NSF, Microsoft Corp., the Alfred P. Sloan Foundation, or the U.S. government.

REFERENCES

- [1] "Cisco Logical Routers," http://www.cisco.com/en/US/docs/ios_xr_sw/iosxr_r3.2/interfaces/command%2Freference/hr32lr.html.
- [2] "Juniper Logical Routers," <http://www.juniper.net/techpubs/software/junos/junos85/feature-guide-85%2Fid-1113921.html>.
- [3] T. Anderson, L. Peterson, S. Shenker, and T. Turner, "Overcoming the Internet Impasse Through Virtualization," in *IEEE Computer*, vol. 38, no. 4, May 2005.
- [4] D. Applegate, G. Galinescu, D. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing Rectilinear Pictures and Minimizing Access Control Lists," in *ACM SODA*, 2007.
- [5] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas: realistic and controlled network experimentation," in *Proc. ACM SIGCOMM*, September 2006.
- [6] Chad R. Meiners and Alex X. Liu and Eric Torng, "Topological Transformation Approaches to Optimizing TCAM-Based Packet Processing System," in *ACM SIGMETRICS*, 2009.
- [7] Cisco, Inc., "Policy-based routing, white paper," http://www.cisco.com/warp/public/732/Tech/policy_wp.pdf.
- [8] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet Classifiers in Ternary CAMs Can Be Smaller," in *ACM SIGMETRICS*, 2006.
- [9] J. Fu and J. Rexford, "Efficient IP-Address Lookup with a Shared Forwarding Table For Multiple Virtual Routers," in *ACM CoNEXT*, 2008.
- [10] M. G. Gouda and A. X. Liu, "Firewall Design: Consistency, Completeness and Compactness," in *Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2004.
- [11] P. Gupta and N. McKeown, "Packet Classification Using Hierarchical Intelligent Cuttings," in *Hot Interconnects*, 1999.
- [12] Juniper Networks, Inc., "Intelligent Logical Router Service," www.juniper.net/solutions/literature/white_papers/200097.pdf.
- [13] P. Lekkas, *Network Processors - Architectures, Protocols, and Platforms*, 2003.
- [14] A. X. Liu, C. R. Meiners, and Y. Zhou, "All-match based complete redundancy removal for packet classifiers in TCAMs," in *IEEE INFOCOM*, 2008.
- [15] C. R. Meiners, A. X. Liu, and E. Torng, "Bit Weaving: A Non-prefix Approach to Compressing Packet Classifiers in TCAMs," in *IEEE ICNP*, 2009.
- [16] T. Mitchell, *Machine Learning*. McGraw Hill, 1997.
- [17] P. Psenak and S. Mirtorabi and A. Roy and L. Nguyen and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF," IETF RFC 4915, 2007.
- [18] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," in *ACM SIGCOMM*, 2003.
- [19] Y. Sung, S. Rao, G. Xie, and D. Maltz, "Towards Systematic Design of Enterprise Networks," in *ACM CoNEXT*, 2008.
- [20] T. Przygienda and N. Shen and N. Sheth, "Multi-Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)," IETF RFC 5120, 2008.
- [21] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," in *ACM Computing Surveys*, vol. 37, no 3, 2005.
- [22] D. E. Taylor and J. S. Turner, "ClassBench: A Packet Classification Benchmark," in *IEEE INFOCOM*, 2005.
- [23] T. Woo, "A Modular Approach to Packet Classification: Algorithms and Results," in *IEEE INFOCOM*, 2000.
- [24] C. Zhang, M. Winslett, and C. Gunter, "On the Safety and Efficiency of Firewall Policy Deployment," in *IEEE Symposium on Security and Privacy*, May 2007.